Universidad de Buenos Aires
Facultad de Ingeniería
Año 2022

86.54 Redes Neuronales
Trabajo final - Clasificación de comandos

Alumna:
Ochagavia, Lara 100637
lochagavia@fi.uba.ar

1. Introducción

El objetivo de este documento es la construcción y análisis de una red neuronal que sea capaz de clasificar 10 comandos de una sola palabra a partir de un *set* de datos de 500 archivos en formato .wav (archivos de audio) con *tensorflow* y *Keras*. Este tipo de redes lleva el nombre de "sistema de detección de palabras clave", o *"keyword spotting system"* en inglés. Al tratar con archivos de audio se realiza un preprocesamiento a los datos para el cual se presenta una breve explicación sobre el análisis del habla. Se incluye, además, una investigación sobre el estado del arte en lo que respecta a la clasificación de comandos en la actualidad.

2. Procesamiento del habla

Existe la necesidad de extraer características de las señales de audio que permitan obtener componentes para identificar contenido relevante. Se busca eliminar información que pueda interferir con el reconocimiento de los datos deseados. Se pensaron, entonces, los coeficientes *cepstrales* en la frecuencia de Mel, o *Mel Frequency Cepstral Coefficients* (MFCCs). Son coeficientes que representan el habla basados en la percepción del oído humano.

Para entender qué son los coeficientes de Mel *Cepstrum*, primero se debe comprender qué son los coeficientes *Cepstrum* por sí solos. El origen de la palabra ***Cepstrum*** es ***Spectrum***, pero se invierten las primeras letras para denotar que se trabaja en el dominio de la transformada de *Fourier* inversa del logaritmo de la transformada de *Fourier*, es decir, en **quefren**cias en lugar de en **frecuen**cias.

$$C(x(t)) = F^{-1}[log(F[x(t)])] \qquad (2.1)$$

Se parte de un pequeño fragmento de la señal temporal de audio y se le aplica la transformada discreta de *Fourier* (DFT), con lo que se obtiene la potencia espectral de la señal. En este paso se podría analizar la presencia de cada componente de frecuencia en la señal temporal. A la potencia espectral se le aplica el logaritmo, lo que permite ver la periodicidad de ciertos armónicos presentes en la señal original. Esta periodicidad hace que sea útil aplicar una transformación para realizar un mejor análisis de la señal, motivo por el cual se aplica la transformada inversa de *Fourier*, la transformada inversa discreta de *Fourier* (IDFT). Los valores *Cepstrum* que se obtienen muestran las componentes **ra**mónicas (proveniente de armónicas) de la señal de audio sometida a las transformaciones mencionadas.

¿Por qué es tan útil obtener los coeficientes *Cepstrum* para el análisis del habla? La respuesta a esta pregunta requiere de conocer un modelo de la producción del habla.

El aparato fonador está compuesto por etapas que cumplen con funciones de respiración, fonación y articulación. El flujo de aire en la expiración es la fuente de energía para la producción de los sonidos. En la laringe se encuentran pliegues vocales que regulan el paso del aire con la apertura y cierre de la glotis. El abrir y cerrarse de la glotis sucede en un ciclo vibratorio con una frecuencia fundamental, o tono glotal, que varía entre 120Hz para hombres y 250Hz para mujeres. La articulación modula la corriente de aire que proviene de la glotis en el tracto vocal, conformado por las cavidades supra glóticas, que actúan como cámaras de resonancia de la excitación glótica. Para pensarlo en términos digitales, el tracto vocal actúa como un filtro que moldea los sonidos de salida del sistema.

El abrir y cerrar de la glotis produce una señal conocida como pulso glótico. Es una señal que puede ser periódica o turbulenta, pero, en cualquier caso, se interpreta como ruido agregado a la señal de audio en el sentido de que entorpece la distinción de parámetros característicos del lenguaje, como lo son los fonemas, dado que solo aporta información sobre el tono del sonido.

Se tiene, entonces, la señal de audio temporal, se le aplica la transformada de *Fourier* y se calcula el logaritmo de ese espectro. En este paso, los máximos de la envolvente de la señal resultante de estas transformaciones presentan información sobre los fonemas conocida como formantes.

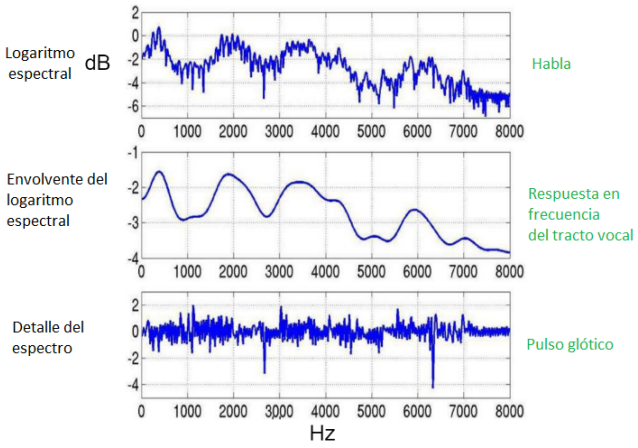

Figura 2.1: Descomposición del logaritmo espectral. Imagen obtenida de (I)

Se puede deducir de lo explicado previamente que la señal de habla es el resultado de convolucionar temporalmente el pulso glótico con la señal del tracto vocal. Modelando matemáticamente al pulso glótico como $e(t)$ y a la señal del tracto vocal

como $h(t)$, se obtiene la señal de habla $x(t)$ como:

$$x(t) = e(t) * h(t). \tag{2.2}$$

Al aplicar la transformada de *Fourier* y posteriormente el logaritmo a la ecuación 2.2 se obtiene:

$$X(t) = E(t) \cdot H(t) \tag{2.3}$$

$$\underbrace{log(X(t))}_{\text{Habla}} = \underbrace{log(E(t))}_{\text{Pulso glótico}} + \underbrace{log(H(t))}_{\text{Tracto vocal}} \tag{2.4}$$

Al aplicar el logaritmo, se logra tratar ambas señales, el pulso glótico y el tracto vocal, de manera separada. Dado que se busca poder descartar la componente de la señal que representa al pulso glótico, se aplica la IDFT para aislar ambas señales. En el dominio de las **quefren**cias se obtiene una señal que no solo se encuentra separada en términos matemáticos, si no que se percibe gráficamente la separación entre ambas componentes.

La señal del tracto vocal se encuentra en la región de bajas **quefren**cias, mientras que la parte relacionada con la excitación glótica se concentra que las **quefren**cias altas. La deconvolución se logra multiplicando al *Cepstrum* por una ventana que separe ambas zonas, es decir, realizando un ***liftering*** (***filtering***, o filtrado, pero en el dominio de *Cepstrum*) de bajas **quefren**cias.

Hasta este momento se explicó lo que es el *Cepstrum*, pero queda comprender lo que son los coeficientes Mel *Cepstrum*. No son otra cosa que los coeficientes *Cepstrum* pero obtenidos a partir del logaritmo del espectro en la escala de frecuencias Mel. Se escala la frecuencia mediante la aplicación de un banco de filtros triangulares solapados y espaciados de forma tal que incorporan factores biológicos de la audición humana, que no percibe las frecuencias en escala lineal, si no que lo hace en escala logarítmica, así como también percibe en dicha escala las amplitudes.

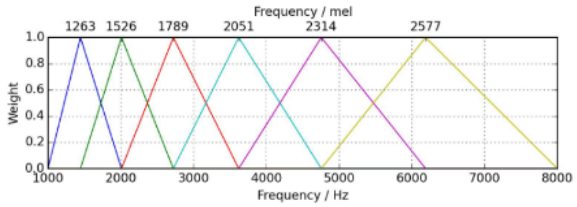

Figura 2.2: Banco de filtros Mel. Imagen obtenida de (I)

Los pasos para obtener los MFCCs son:

- Aplicar la DFT de la señal temporal
- Aplicar el logaritmo a la amplitud del espectro
- Escalar la frecuencia con el banco de filtros triangulares
- Aplicar la transformada discreta de *Fourier* del coseno

Se observa que en lugar de aplicar la transformada inversa de *Fourier*, se aplica la transformada discreta de *Fourier* del coseno. Esta simplificación se debe a que la transformada discreta del coseno devuelve valores de coeficientes reales, permite decorrelacionar la energía en bandas de Mel y reduce la cantidad de dimensiones necesarias para representar el espectro. Los primeros coeficientes son los que contienen mayor información, es decir, la información relevante para el procesamiento del habla. Como estándar, se aplican 24 filtros Mel y se extraen los primeros 13 coeficientes.

3. Red convolucional - Audio como imágenes

Las redes convolucionales son un tipo de algoritmo de *deep learning* que funciona muy bien para el aprendizaje de imágenes. Esto se debe a su capacidad de aprender patrones invariantes a la traslación y que tienen jerarquía espacial. La interpretación humana de las imágenes sucede de forma similar a lo que lo hace una red convolucional.

Lo que da el nombre a este tipo de redes neuronales es la convolución. En el contexto de *deep learning* la convolución se lleva a cabo a partir de una cuadrícula cuyos casilleros contienen valores llamados pesos. Dicha cuadrícula lleva el nombre de *kernel* y se le 'aplica' a la imagen. Con la palabra 'aplica' se hace referencia a que se realiza la convolución entre el *kernel* y la imagen, que se representa en una grilla en la cual cada casillero representa un píxel cuyo valor puede variar entre 0 y 255, en caso de ser una imagen en blanco y negro (en imágenes a color cada píxel se representa con un vector de tres valores entre 0 y 255). La convolución sigue la fórmula que se presenta a continuación:

$$\sum_{i=1}^{P} = \text{imagen}_i \cdot \text{kernel}_i. \tag{3.1}$$

Los *Kernel* sirven para detectar características de las imágenes, como pueden ser bordes o líneas, diferentes orientaciones de las mismas, como ejemplo. Los valores de estas grillas, sin embargo, no se definen manualmente, si no que la red neuronal los define en el proceso de aprendizaje siguiendo ciertas reglas impuestas para obtener el mejor resultado posible.

Cuando se define una red neuronal convolucional se tienen que tomar decisiones sobre su arquitectura. Estas decisiones consisten en definir la

dimensión de las entradas, la dimensión de las grillas de pesos, el desplazamiento de estas grillas sobre la grilla de píxeles, la profundidad de los *Kernel* (varía si se trabaja con imágenes a color o en blanco y negro) y el número de *Kernels* que se utilizarán. Cada capa convolucional de la red puede tener múltiples *Kernels*, pero cada *Kernel* devuelve como resultado una grilla de dos dimensiones, por lo que la salida de la capa convolucional será de tantos arreglos de dos dimensiones como *Kernels* se utilicen.

Las redes convolucionales, además de convolucionar las grillas de píxeles con las grillas de pesos, realizan lo que se conoce como *pooling*. Se realiza la reducción de resolución de las imágenes, para lo cual se deben definir el tamaño de la grilla que realiza esta reducción, el desplazamiento de la grilla sobre la imagen, y el tipo de *pooling* que se aplicará, por ejemplo: *max pooling, min pooling* o *average pooling*. El método de *average pooling* suaviza la imagen, por lo que las características bruscas pueden no ser identificadas al aplicar este tipo de reducción; El método *max pooling* selecciona los píxeles más brillantes de la imagen y lo contrario sucede si se selecciona *min pooling*.

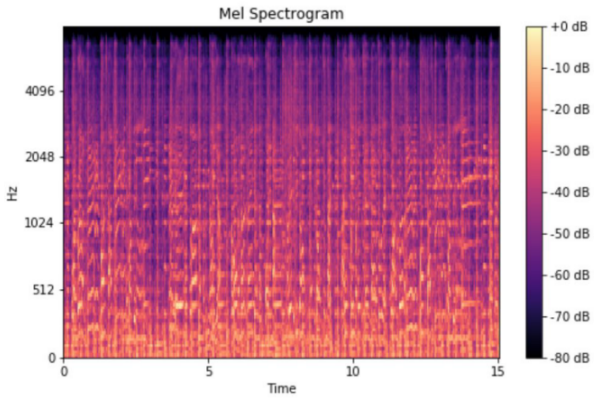

Figura 3.1: Visualización de los MFCCs. Imagen obtenida de (I)

En el presente trabajo no se está tratando con imágenes, sino que se está tratando con archivos de audio, de los cuales se extraen los MFCCs. ¿Cómo se pueden aplicar los coeficientes Mel *Cepstrum* a una red convolucional? Los MFCCs pueden interpretarse como imágenes, es decir, el tiempo y la frecuencia pueden interpretarse como las coordenadas de los eje de abscisas y ordenadas de una imagen, y la amplitud del coeficiente puede interpretarse como el valor del píxel. Las imágenes tienen una tercera dimensión que indica su color y puede ser escala de grises o RGB. En audio se suele usar un solo canal, indicando que se lo trata como escala de grises.

4. Preprocesamiento de los datos

Como ya se mencionó en la introducción, se utiliza un *set* de 500 audios, 50 archivos de audio por cada uno de los 10 comandos a clasificar. Los audios se extrajeron del *dataset* de *Google* llamado *Speech Commands Dataset* (set de datos de comandos de voz).

Para preparar el *set* de datos en el formato que servirá para la implementación de la red neuronal de este trabajo, se creará un archivo *.json* en el cual se guardará la información procesada, lista para utilizarse como entrada de la red. Como se trabaja con archivos y específicamente con el formato *.json* se importan en *python* el módulo *os* y el paquete *json*.

Se busca extraer los mencionados coeficientes Mel *Cepstrum* de las señales de audio, para lo cual se utiliza el paquete de *python* llamado *librosa*, un paquete que provee las herramientas necesarias para el procesamiento de audio.

Previo a comenzar con el procesamiento de los audios, se realiza un mapeo de las posibles palabras que la red podrá distinguir de los audios y a partir de las cuales se les aplica una etiqueta a los archivos según la palabra que contengan. Las palabras seleccionadas son:

- yes
- go
- left
- on
- up
- no
- stop
- right
- off
- down

Los audios disponibles en el *set* de datos tienen una duración promedio de un segundo, y en *librosa* se tiene como estándar que un segundo de señal de audio equivale a 22050 muestras. Si se encuentra algún audio con una duración mayor a un segundo, se recorta para que su duración sea de exactamente un segundo.

Los MFCCs se extraen con la función *librosa.feature.mfcc*, l cual recibe como parámetros: la señal de la cual se busca extraer los coeficientes, la cantidad de coeficientes a extraer, la longitud de los segmentos iguales en los que se dividirá la señal temporal y la ventana para la transformada de *Fourier*. Para la implementación de este trabajo se fijaron los siguientes valores: 13 coeficientes Mel *Cepstrum*, segmentos de la señal de 512 muestras y una ventana para la transformada de *Fourier* de valor 2048.

Con este procesamiento, se tienen como resultado muestras cuyas dimensiones resultan de $(22050/512, 13) = (44, 13)$. Para recopilar, se tienen 22050 muestras por audio en segmentos de 512 muestras, 13 MFFCs. Se deberá agregar una tercera dimensión que es la adaptación del color de las imágenes al audio, la cual suele fijarse en 1 e

indica que se trata como una imagen en blanco y negro.

5. Modelo de red

La red neuronal implementada consiste de tres capas convolucionales seguidas por la etapa de clasificación, compuesta por capas '*flatten*', '*dense*' y '*dropout*'.

```
#1ra capa convolucional
model.add( Conv2D(64, (3,3), activation="relu",
                  input_shape = input_shape))
model.add( BatchNormalization() )
model.add( MaxPool2D((3,3), strides = (2,2),
                     padding = "same") )

#2da capa convolucional
model.add( Conv2D(32, (3,3), activation="relu"))
model.add( BatchNormalization() )
model.add( MaxPool2D((3,3), strides = (2,2),
                     padding = "same") )

#3ra capa convolucional
model.add( Conv2D(32, (2,2), activation="relu"))
model.add( BatchNormalization() )
model.add( MaxPool2D((2,2), strides = (2,2),
                     padding = "same") )

#'Flatten' -> pasa la salida a un arreglo unidimensional
#             para 'alimentar' una 'dense layer'
model.add( Flatten() )
model.add( Dense(64, activation="relu") )
model.add( Dropout(0.3) )

#Clasificador con función de activación 'softmax'
model.add( Dense(NUM_KEYWORDS, activation="softmax") )
```

Figura 5.1: Arquitectura de la red implementada

5.1. Función de activación

La función de activación le da la habilidad al modelo de agregar no-linealidad. Se utiliza la función *RELU* que pone en 0 los pesos negativos y mantiene el valor de los pesos no negativos.

La función de activación de la última capa implementada es la función *softmax*, la cual devuelve como salida la probabilidad de cada clase, por lo que sirve para el objetivo de clasificar.

5.2. Optimizador y función de pérdida

Se utiliza el optimizador *Adam*, el cual controla la tasa de aprendizaje. La función de pérdida se usa para evaluar qué tan diferente es la predicción de la información real y penaliza al modelo cuando la predicción no es buena. Se utiliza en este caso la función de pérdida *Sparce Categorical Cross-Entropy*, la cual se aplica en tareas de clasificación multiclase, es decir, cuando se tiene múltiples muestras pero cada una se corresponde con una sola clase.

El objetivo es minimizar la función de pérdida. A menor pérdida, mejor el modelo. La entropía de una variable aleatoria X es el nivel de incertidumbre inherente en el posible resultado de la variable. A mayor entropía, mayor incertidumbre de la distribución de probabilidad y a menor entropía menor incertidumbre. La función de pérdida *Cross-Entropy* compara la probabilidad de cada predicción con la verdadera clase y se calcula un valor de pérdida que penaliza la probabilidad según qué tan lejos se encuentre del valor real esperado. Esta penalización es logarítmica, lo que lleva a grandes penalizaciones cuando las diferencias son cercanas a 1 y a pequeñas penalizaciones cuando las diferencias resultan cercanas a 0. *Sparse Categorical Cross-Entropy* se diferencia de *Cross-Entropy* únicamente por la forma en que están codificadas las etiquetas, que es en forma de enteros.

6. Entrenamiento y testeo

Se obtienen los datos a partir del *set* guardado y preprocesado en el archivo *.json* mencionado con anterioridad. Se tienen 21312 *features*, resultado de tomar intervalos de 512 muestras de cada uno de los 5000 audios. Se toma un 10 % de esas *features* para utilizar como datos para el testeo, y el 90 % restante se utiliza para realizar el entrenamiento. Del 90 % destinado a entrenar a la red, se toma el 10 % para utilizar como *set* de validación, es decir, se destina un 9 % de las *features* para validar. El *set* de validación se utiliza para ajustar manualmente los hiper-parámetros de la red ante los resultados que se devuelvan por el entrenamiento realizado.

A todas las *features* se les agregó la tercera dimensión con valor 1, indicando un canal en blanco y negro para la adaptación de las dimensiones de los MFCCs a la entrada de una red convolucional, resultando en arreglos de dimensiones (44,13,1).

```
#Entrenamiento de la red
hist = model.fit(X_train, y_train,
          epochs = EPOCHS,
          batch_size = BATCH_SIZE,
          validation_data = (X_validation, y_validation)
          )

#Evaluación de la red
test_error, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test error: {test_error}, test accuracy: {test_accuracy}")
```

Figura 6.1: Código implementado para entrenar y posteriormente evaluar la red.

Los parámetros '*EPOCHS*' y '*BATCH_SIZE*' que pueden observarse en la figura 6.1 se fueron

modificando hasta obtener lo que se considera como el mejor resultado de la red.

7. Resultados

Se comienza entrenando a la red con un *batch size* de 128 y sólo 5 *epochs*. Con estos parámetros se obtuvo:

Epochs = 5, Batch size = 128		
	Loss	Accuracy
Train	1,2394	57,21 %
Validation	1,1219	63,82 %
Test	1,1008	66,46 %

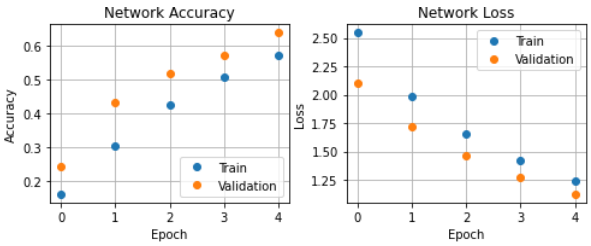

Figura 7.1: Epochs=5, Batch size=128.

Para mejorar los resultados, se disminuyó el *batch size* a 64 y se aumentó el valor de *epochs* a 10, obteniéndose así:

Epochs = 10, Batch size = 64		
	Loss	Accuracy
Train	0,6292	78,94 %
Validation	0,5360	83,00 %
Test	0,5623	81,61 %

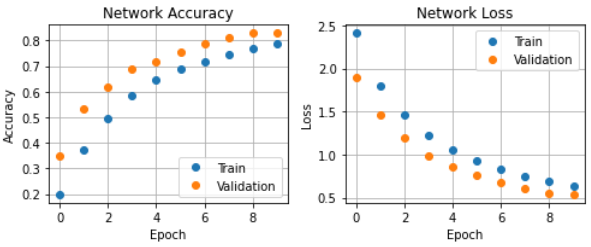

Figura 7.2: Epochs=10, Batch size=64.

Dada la evidente mejora en la precisión, se mantuvo el valor del *batch size* pero se duplicó el valor de *epochs*. Se obtuvo con los nuevos parámetros:

Epochs = 20, Batch size = 64		
	Loss	Accuracy
Train	0,3488	88,77 %
Validation	0,3234	90,09 %
Test	0,3486	88,78 %

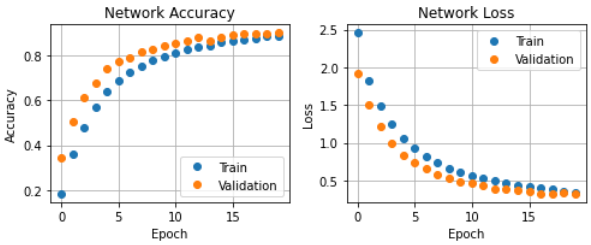

Figura 7.3: Epochs=20, Batch size=64.

Se continuó incrementando el número de *epochs* a 30 y se mantuvo el valor del *batch size* en 64, resultando:

Epochs = 30, Batch size = 64		
	Loss	Accuracy
Train	0,2364	92,28 %
Validation	0,2917	90,46 %
Test	0,2658	91,46 %

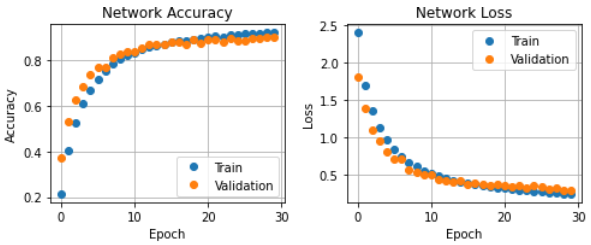

Figura 7.4: Epochs=30, Batch size=64.

Con los últimos parámetros se obtiene un muy buen resultado, con un error muy pequeño y un alto porcentaje de precisión, pero se continua buscando mejorar los resultados. Con un valor de *epoch* de 40 y un *batch size* de 64 se obtiene:

Epochs = 40, Batch size = 64		
	Loss	Accuracy
Train	0,1857	93,93 %
Validation	0,2638	91,14 %
Test	0,2553	91,65 %

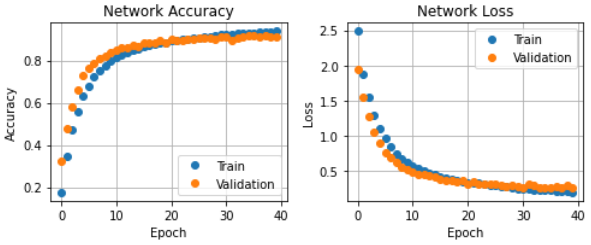

Figura 7.5: Epochs=40, Batch size=64.

Las mejoras no resultan muy evidentes con la última modificación, por lo que se procede a disminuir a la mitad el tamaño del *batch size*, fijando

su valor en 32, y manteniendo el valor de *epochs* en 40:

Epochs = 40, Batch size = 32		
	Loss	Accuracy
Train	0,1777	94,15 %
Validation	0,2571	91,61 %
Test	0,2018	93,62 %

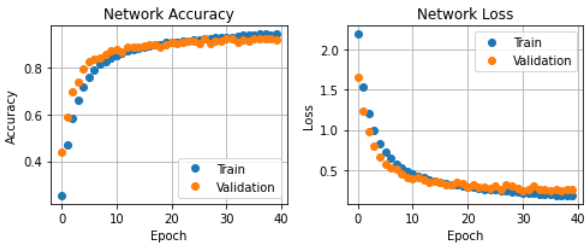

Figura 7.6: Epochs=40, Batch size=32.

Se puede analizar que la arquitectura de red adoptada para el problema de clasificación multiclase resulta adecuada y se logró una *accuracy* superior al 90 % para la clasificación de las muestras apartadas en el *set* de testeo. A modo de ilustración, se presentan a continuación las primeras 15 predicciones y sus etiquetas correspondientes:

```
0)
Clase original:  off
Predicción:  off : 99.9915 %
1)
Clase original:  left
Predicción:  left : 99.9217 %
2)
Clase original:  stop
Predicción:  stop : 100.0 %
3)
Clase original:  yes
Predicción:  yes : 99.9935 %
4)
Clase original:  down
Predicción:  down : 99.9981 %
5)
Clase original:  right
Predicción:  right : 99.9495 %
6)
Clase original:  off
Predicción:  off : 58.8001 %
7)
Clase original:  yes
Predicción:  yes : 99.9998 %
8)
Clase original:  down
Predicción:  down : 75.816 %
9)
Clase original:  right
Predicción:  right : 99.9992 %
10)
Clase original:  stop
Predicción:  stop : 99.9972 %
11)
Clase original:  right
Predicción:  right : 99.9997 %
12)
Clase original:  yes
Predicción:  yes : 100.0 %
13)
Clase original:  on
Predicción:  on : 100.0 %
14)
Clase original:  stop
Predicción:  stop : 99.9945 %
```

8. Estado del arte

Para la mayoría de las aplicaciones de audio en *deep learning* se utilizan espectrogramas (o los espectrogramas con coeficientes de Mel) para representar al audio. Se comienza con la señal de audio temporal, se toman muestras y se aplican las transformaciones matemáticas necesarias para obtener el espectrograma, a los cuales se les pueden aplicar procedimientos para aumentar la información que representan. Se utilizan arquitecturas de redes convolucionales estándares para procesar la *data* y extraer el 'mapeo' de características contenidas en la representación del espectrograma. El siguiente paso resulta ser la generación de las predicciones de la información codificada, que depende específicamente del problema particular para el cual se esté aplicando *deep learning*.

Dada la fuerte presencia de sonidos en la vida cotidiana y el rango de tipos de sonidos existentes, existe un amplio rango de aplicaciones que requieren del procesamiento y análisis del audio, para lo cual resulta útil aplicar *deep learning*. Se presentan algunas a continuación:

Separación y segmentación de audio Para la separación se audio se requiere aislar una señal de interés de un grupo de señales tal que se la pueda usar para posterior análisis. Se puede querer aislar el sonido de la voz de una persona del ruido ambiente del audio o el sonido de algún instrumento particular del resto de la música. La segmentación de audio se usa para resaltar ciertas secciones relevantes del audio. Se puede usar, por ejemplo, para detectar los diferentes sonidos presentes en ciertos procesos biológicos y poder detectar anomalías.

Clasificación y etiquetado de géneros musicales Al día de hoy existen múltiples plataformas que ofrecen servicios transmisión de música, y para hacerlo, suelen utilizar clasificadores de audio para categorizar el género musical basado en el audio. Se analiza el contenido de la música para clasificar con cuál o cuáles géneros coincide, lo que lo hace un problema conocido como *multi-label*. Este tipo de aplicaciones no solo abarca el procesamiento del audio, si no que también se trabaja con información sobre la propiedad del audio, la fecha de lanzamiento, y otros datos, que pueden hacer más detallista a la clasificación.

Generación y transcripción musical Existe la posibilidad de generar audio musical que se corresponda con géneros particulares o algún instrumento. La transcripción musical permite realizar el proceso inverso, es decir, crear una partitura con las notas presentes

en cierto contenido musical.

Clasificación de audios y reconocimiento de voz Este problema consiste en tomar sonidos y asignarlos a una de varias clases posibles. La tarea puede ser la identificación del tipo o la fuente del sonido. Una posible aplicación para la cual la clasificación de audios puede resultar útil es la detección de fallas en sistemas mecánicos, por ejemplo.

El reconocimiento de voz es un problema de clasificación, pero trata con reconocer sonidos del habla. Se puede reconocer desde el género que generó el sonido, la persona particular que haya producido el audio, se pueden detectar incluso emociones o simplemente clasificar qué es lo que se dijo. Esta última aplicación lleva a otro área:

Diálogo a texto y texto a diálogo Se pueden reconocer las palabras presentes en un audio de voz. Esta tarea abarca extraer las palabras del audio en el lenguaje en el que se encuentren y transcribirlas en texto. No sólo implica la dificultad del análisis de los audios, si no que también se tiene el desafío de aprender ciertas características o reglas de los lenguajes. Con la síntesis del habla se puede realizar el camino inverso y generar diálogos a partir de texto. Los ejemplos más conocidos de este tipo de aplicación son los asistentes virtuales. Los mencionados asistentes virtuales cuentan con la tecnología de reconocimiento automático del habla, o *automatic speech recognition* (ASR).

9. Bibliografía

I. Velardo, V. (2021). Mel-Frequency Cepstral Coefficients Explained Easily [Diapositivas]. GitHub. https://github.com/musikalkemist/AudioSignalProcessingForML/blob/master/19-%20MFCCs%20Explained%20Easily/Mel-Frequency%20Cepstral%20Coefficients%20Explained%20Easily.pdf

II. Velardo, V. (2020). How does a convolutional neural network work [Diapositivas]. GitHub.https://github.com/musikalkemist/DeepLearningForAudioWithPython/blob/master/15-%20How%20does%20a%20convolutional%20neural%20network%20work/slides/How%20does%20a%20convolutional%20%20neural%20network%20work.pdf

III. Nandi, P. (2022, 7 enero). CNNs for Audio Classification - Towards Data Science. Medium. https://towardsdatascience.com/cnns-for-audio-classification-6244954665ab

IV. What is validation data used for in a Keras Sequential model? (2017, 19 septiembre). Stack Overflow. https://stackoverflow.com/questions/46308374/what-is-validation-data-used-for-in-a-keras-sequential-model

V. Koech, K. E. (2021, 16 diciembre). Cross-Entropy Loss Function - Towards Data Science. Medium. https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

VI. Doshi, K. (2022, 7 enero). Audio Deep Learning Made Simple: Sound Classification, Step-by-Step. Medium. https://towardsdatascience.com/audio-deep-learning-made-simple-sound-classification-step-by-step-cebc936bbe5

VII. Doshi, K. (2021, 30 diciembre). Audio Deep Learning Made Simple (Part 1): State-of-the-Art Techniques — by Ketan Doshi — Towards Data Science. Medium. https://towardsdatascience.com/audio-deep-learning-made-simple-part-1-state-of-the-art-techniques-da1d3dff2504